1 — **Link Design**

Communication Link
Requirements

subsystem model
descriptions

2 — **Algorithm Design**

Tx → Chan → Rx

3 — **Architecture Design**

MAP
DF ⊗
DF ⊗

4 — **Circuit Design**

Bit Error Rate
Performance Estimation

Aqcuisition Performance
Tracking Stability
Spectral Properties
Multiaccess

Active Area
Throughput
Estimated Power
Estimated Timing
Bittrue behavior

Power
Routing Capacitance
Netlist Errors

FIG. 1A

FIG. 1B

APPROVED	O.G. FIG.	
	CLASS	SUBCLASS
BY		
DRAFTSMAN		
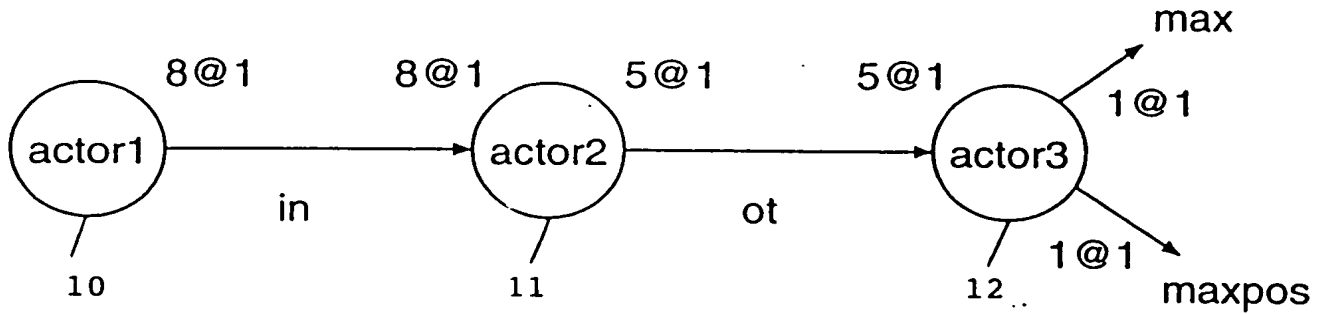
FIG. 2
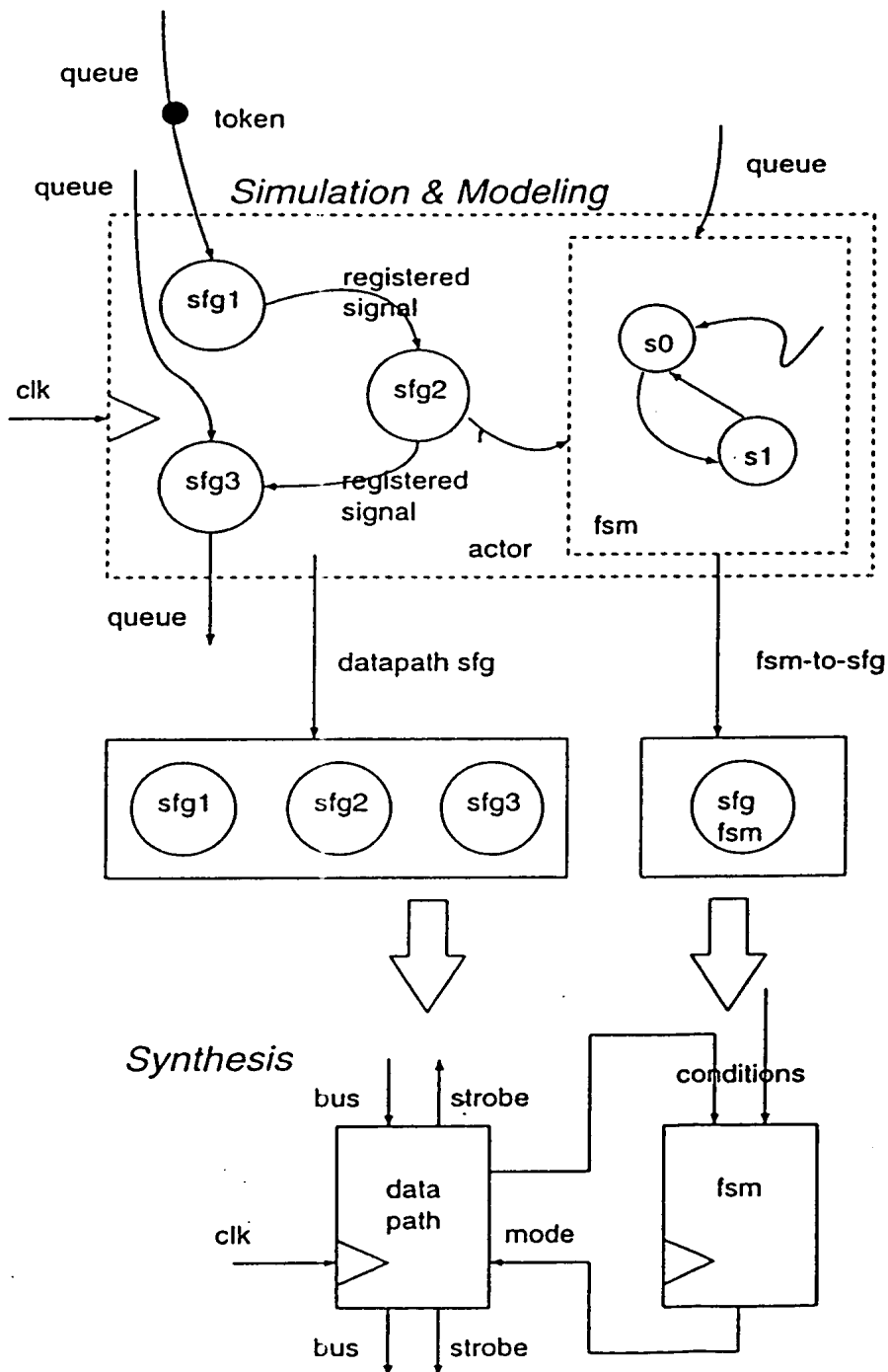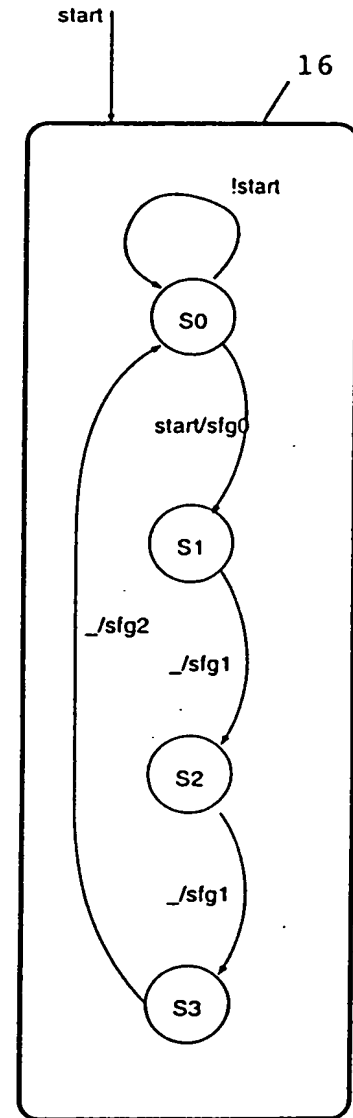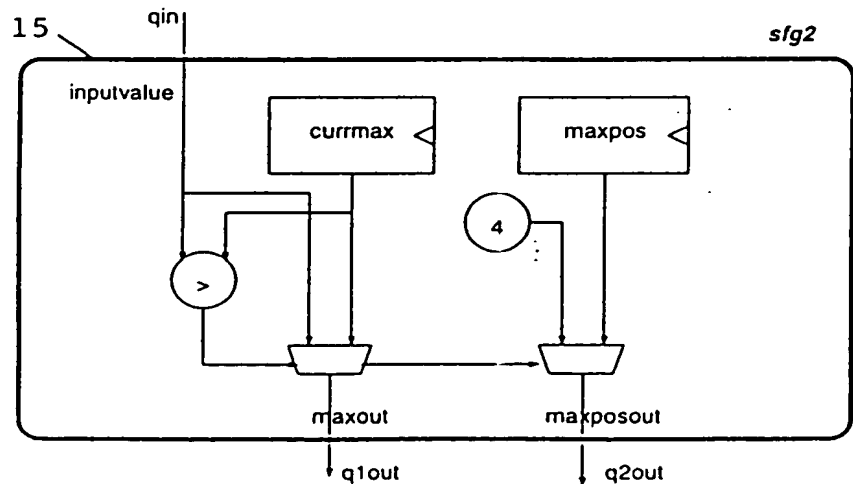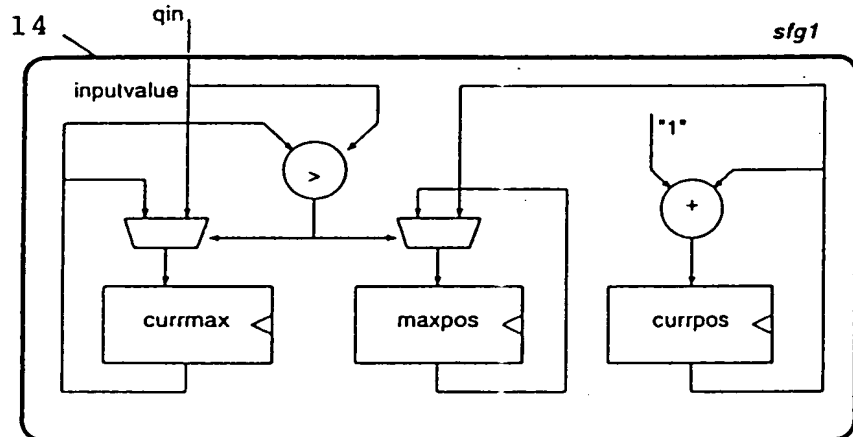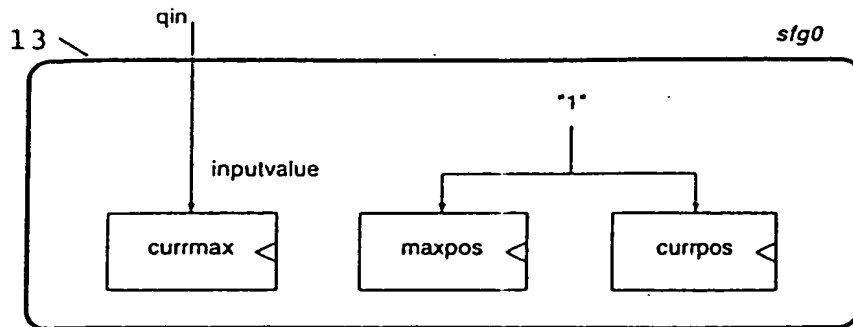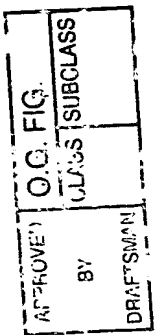
FIG. 3

FIG. 4

FIG. 5

max

8@1   8@1   5@1   5@1   1@1

actor1 → actor2 → actor3

in   ot   1@1

10   11   12   maxpos

FIG. 6

FIG. 7

FIG. 8

travel delay = 5
token concurrency = 1
token latency = 5

each 5 cycles
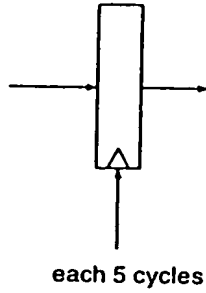
travel delay = 5
token concurrency = 1
token latency = 1

each cycle
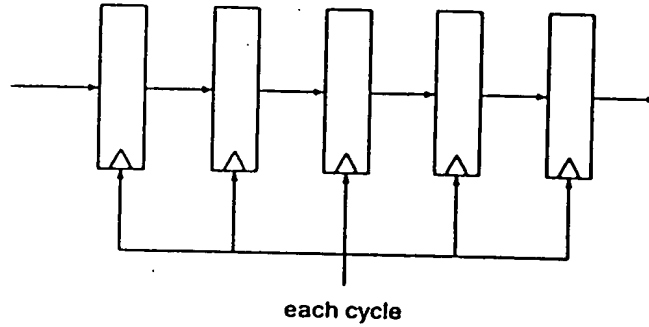
travel delay = 5
token concurrency = 2
token latency = 1

each cycle

FIG. 9

token age   queue age

actor → ● → | queue | →

token

iteration delay   travel delay

b
a → (+) → c

operation

if actor fires:

    for all actor output queues:

        queue age = queue age + iteration delay

---

if operator is executed:

    output token age = max(input token ages)

if token enters queue:

    token age = token age + travel delay

CHECK EXPECTATION:

    token age at output of queue

            > queue age

IF SUCCESS:

    queue age = token age

IF FAIL:

    warn user

    token age = queue age

FIG. 10

```
dfix T_sample(8, 6);
dfix T_acc   (8, 6);
dfix T_bit   (1, 0, ns);
double hardwired_coef = { 0.5, 0.2, -0.3, 0.15 };

fsm correlator::define(clk & _ck)
{
  sig_array  coef       (4, ck, T_sample);
  sig_array  sample     (4, ck, T_sample);
  sig        accu       (ck, T_accu  );
  sig        sample_in (T_sample     );
  sig        coef_in   (T_sample     );
  sig        corr_out  (T_sample     );
  sig        load      (ck, T_bit    );
  sig        load_ctr  (T_bit        );

  sfg initialize_coefs;
  for (i = 0; i < 4; i++)
     coef[i] = W(T_sample, hardwired_coef[i] );

  sfg load_coef_0;
  input(coef_in);
  coef[0] = in_coef_in;

  sfg correl_1;
  accu   = cast(T_acc, coef[0] * sample[0] + coef[1] * sample[1]);

  sfg correl_2;
  corr   = accu + cast(T_acc,  coef[2] * sample[2] + coef[3] * sample[3] );
  output(corr);

  sfg read_sample;
  input(sample_in);
  for (i = 3; i >=0; i--)
     if (i)
       sample[i] = sample[i-1];
     else
       sample[i] = sample_in;

  sfg read_control;
  input(load_ctr);
  load = load_ctr;

  fsm myfsm;
  initial rst;
  state phase_1;
  state phase_2;
  rst    << always  << initialize_coefs       << phase1;
  phase1 << always  << read_control
                    << correl_1                << phase2;
  phase2 << !cnd(load) << correl_2
                    << read_sample             << phase1;
  phase2 << cnd(load) << correl_2
                    << read_sample
                    << load_coef_0             << phase1;
  return myfsm;
}
```
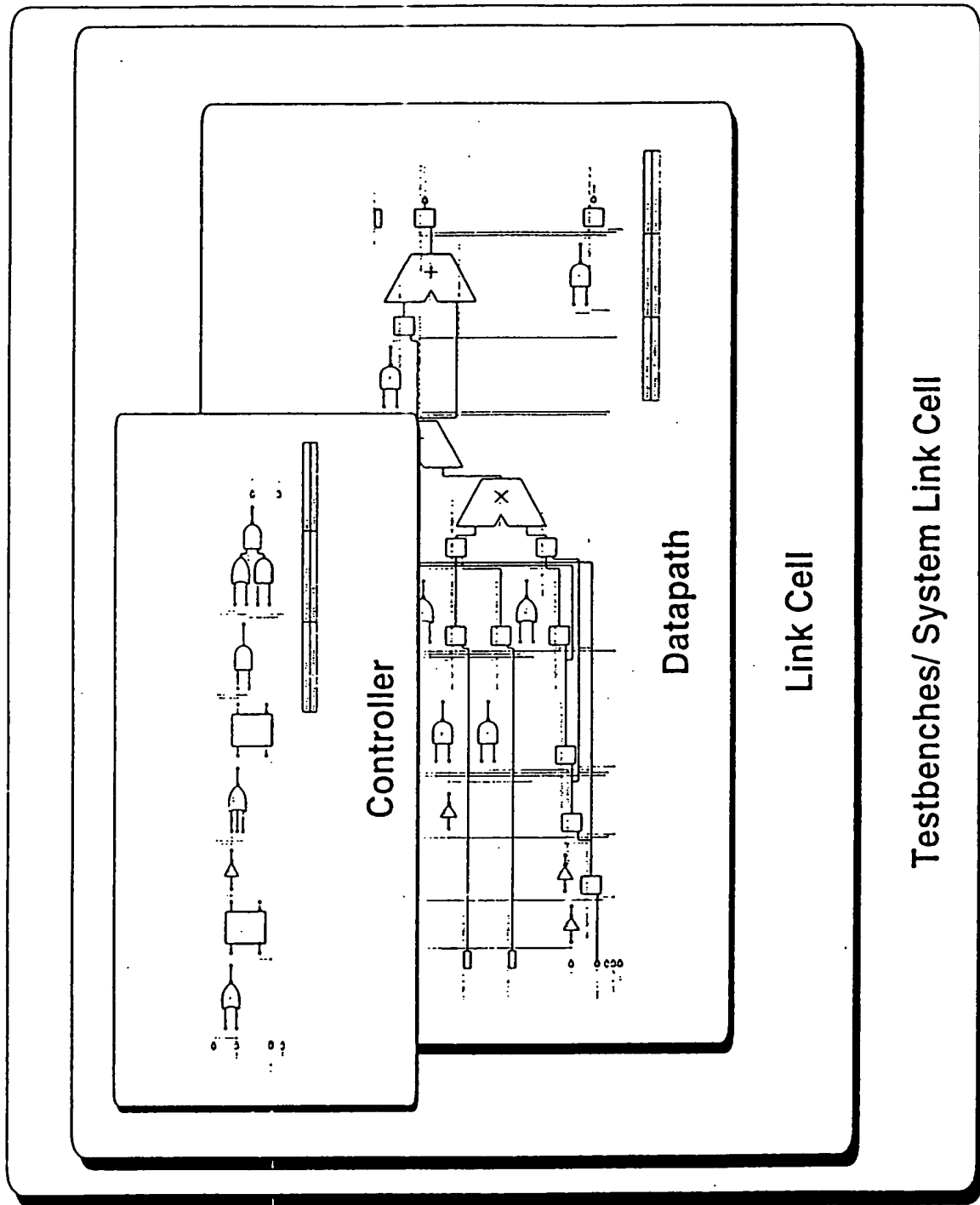
FIG. 11

FIG. 12

FIG. 13

FIG. 14

reset processing

30

!hold_request /
lookup

hold_request /
hold_on

hold_request /
wait_lookup

execute ⟷ hold

!hold_request /
hold_lookup

SFG lookup

SFG hold_on

31

PC → lookup Table → datapath instruction busses

hold_request

hold_pc

PC

hold_request

SFG wait_lookup

"nop"
"nop"
"nop"

→ datapath instruction busses

hold_request

SFG hold_lookup

hold_pc → lookup Table → datapath instruction busses

hold_request

FIG. 15

15.

Sig Class

```
class sig {
  Value value;
  char  *name;
public:
  sig(Value v);
  sig operator +(sig v);
  virtual Value simulate();
  virtual void gen_code(ostream &os);
};

sig sig::operator +(sig v) {
  sigadd s;
  add.left  = &v;
  add.right = this;
  return add;
}

Value sig::simulate() {
  return value;
}

sig::gen_code(ostream &os) {
  os << name;
}
```
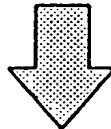
Derived Operator Class

```
class sigadd : public sig {
  sig *left;
  sig *right;
public:
  Value simulate();
   void gen_code(ostream &os);
};

Value sigadd::simulate() {
  return left->eval() +
         right->eval();
}

sigadd::gen_code(ostream &os) {
  os << left->cg()
     << " + "
     << right->cg();
}
```

FIG. 16

```
sig a, b, c, d;
b = a + 3;
d = (b + c) << 3;
```

40
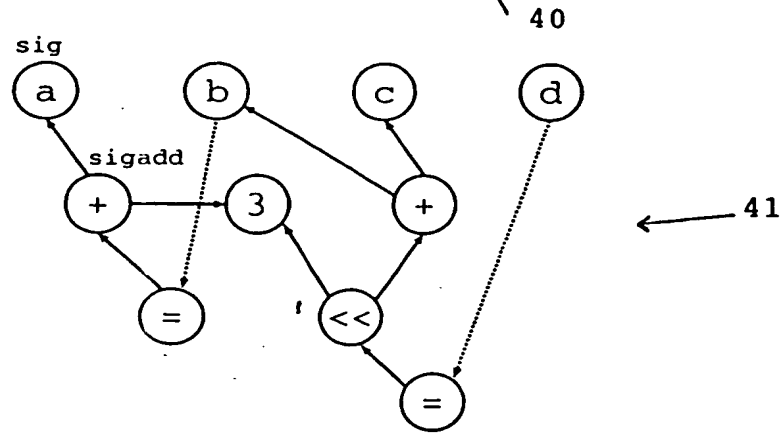
FIG. 17
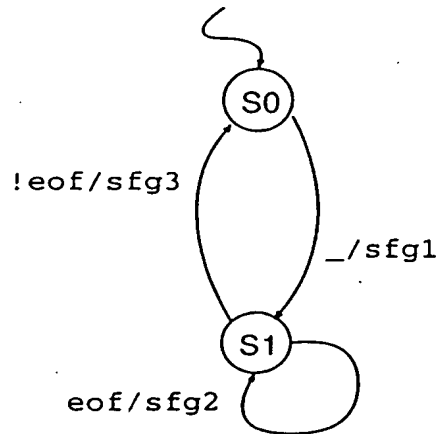

```
fsm f;
initial s0;
state s1;

s0 << allways    << sfg1 << s1;
s1 << cnd(eof)   << sfg2 << s1;
s1 << !cnd(eof)  << sfg3 << s0;
```

FIG. 18

FIG. 19

to
equalizer

found_max

data

HDET
FSMD

header detector

data_present
(condition)

data

HCOR
FSMD

header correlator

from
A/D

FIG. 20

One Clock Cycle

token production | evaluation | register update

clk

60 **Component 1**

FSM

SFG

1

Component 2 61

FSM

+

1   SFG

2a

2b

System Interconnect

Component 3 62

```
int c::run() {
  // firing rule
  ...
  // behavior
  ...
}
```

FIG. 21

```
        ┌─────────────────────┐
        │   C++ High Level    │
        │  System Description  │
        └─────────────────────┘
                   │
               Refinement
                   │
                   ▼
        ┌─────────────────────┐
        │  C++ Timed, Bittrue │
        │  System Description  │
        └─────────────────────┘
                   │
               Compile
               Execute
                   │
                   ▼                   Interpreted
        ┌─────────────────────┐        Simulation
        │        FSMD         │──┐
        │    Data Structure    │  │
        └─────────────────────┘
          C++                  HDL
     Code Generation      Code Generation
         │                      │
         ▼                      ▼
  ┌──────────────┐      ┌──────────────────┐
  │   C++ RT     │      │  HDL Component   │
  │ Description  │      │   Descriptions    │
  └──────────────┘      └──────────────────┘
         │                      │       HDL
      Compile                   │   Simulation &
         │                      │     Synthesis
         ▼                      ▼
  ┌──────────────┐      ┌──────────────────┐
  │Compiled Code │      │    Hardware      │
  │  Simulation  │      │ Implementation   │
  └──────────────┘      └──────────────────┘
```

FIG. 22

Figure 23.

Figure 24.